Kubernetes Network Custom Resource Definition De-facto Standard

Network Plumbing Working Group

Version 1.1

0. Document Versioning
This document shall be versioned in the MAJOR.MINOR format with an optional “-dev” suffix. The “-dev” suffix denotes that the document of the given version is under modification and those changes from the previous version have not yet been ratified by the Group. It is not recommended to rely on a document version with the “-dev” suffix as it may change.

For example, a version of “1.2-dev” means that changes are being made to the document and the next official version of the document will be 1.2. A version of “1.2” means that the any changes since the previous version have been ratified by the Group and the document is ready for implementations to rely upon. Ratified versions are always tagged in Google Docs and also stored in the Group’s github repository:
https://github.com/k8snetworkplumbingwg/multi-net-spec

1. Goals
This document proposes a specification of the requirements and procedures for attaching Kubernetes pods to one or more logical or physical networks, including requirements for plugins using the Container Network Interface (CNI) to attach pod networks.

1.1 Non-Goals of Version 1
This document specifically does not address certain issues for reasons of simplicity and/or the need to achieve some reasonable consensus. These issues may be addressed in future versions of this specification.

1.1.1 Scheduling and resource management
Efforts to integrate scheduling and resource management (eg ensuring that a node is not assigned more pods than there are network resources available for) are underway in the Resource Management Working Group in which various members of this working group and SIG Network are involved. Future versions of this specification may incorporate aspects of resource management.

1.1.2 Changes to the Kubernetes API
This specification explicitly avoids changes to the existing Kubernetes API. Those changes require a much longer upstream process, but the hope is that this document can serve as a

basis for those changes in the future by proving the concepts and providing one or more real-world implementations of multiple pod network attachments.

1.1.3 Interaction with the Kubernetes API

The interaction of additional pod network attachments with the Kubernetes API and its objects, such as Services, Endpoints, proxies, etc is not specified. SIG Network discussed this topic at length in July/August 2017 and decided it would require changes to the Kubernetes API, which is an explicit non-goal of this specification.

1.1.4 Changes to the Kubernetes CNI Driver

To ensure easy use of this specification and implementations of it, no changes will be required of the Kubernetes CNI driver (eg pkg/kubelet/dockershim/network/cni/*). Changes to the driver to support multiple pod network attachments have already been proposed and multiple proof-of-concepts written, but there is as yet no upstream consensus on how multiple attachments should work and thus what changes would be required in the CNI driver.

1.1.6 Enabling Implementations That Are Not CNI Plugins

This specification only attempts to enable Kubernetes network plugins which use the CNI driver of a Kubernetes runtime like dockershim, CRI-O, and rkt. This specification expects pod network attachment operations to be directed through the implementation, which is required to be a CNI plugin. Future versions of this specification may revisit this requirement.

2. Definitions

2.1 Implementation

The Kubernetes network plugin that implements this specification; as of Kubernetes 1.11 this is defined to be a plugin conforming to the Container Network Interface (CNI) specification v0.1.0 or later. Kubernetes should be configured to call the implementation for all pod network operations, and the implementation then determines which additional operations to perform based on the pod's annotations and the Custom Resources defined in this specification.

2.2 Kubernetes Cluster-Wide Default Network

A network to which all pods are attached following the current behavior and requirements of Kubernetes.

2.3 Network Attachment

A means of allowing a pod to directly communicate with a given logical or physical network. Typically (but not necessarily) each attachment takes the form of a kernel network interface placed in to the pod's network namespace. Each attachment may result in zero or more IP addresses being assigned to the pod.

2.4 CNI Delegating Plugin

An implementation conforming to this specification which delegates pod network attachment/detachment operations to other plugins conforming to the CNI specification. Examples include Multus and CNI-Genie.

This specification places CNI Delegating Plugin specific requirements and recommendations under sections marked as such. If the implementation is not a CNI Delegating Plugin, it is free to ignore the requirements and recommendations in such marked sections, and any subsections of them.

3. NetworkAttachmentDefinition Object

This specification defines a `NetworkAttachmentDefinition` Custom Resource object which describes how to attach a pod to the logical or physical network referenced by the object.

3.1 Custom Resource Definition (CRD)

The CRD tells Kubernetes API how to expose `NetworkAttachmentDefinition` objects. See below for the definition of the object itself.

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: network-attachment-definitions.k8s.cni.cncf.io
spec:
  group: k8s.cni.cncf.io
  version: v1
  scope: Namespaced
  names:
     plural: network-attachment-definitions
     singular: network-attachment-definition
     kind: NetworkAttachmentDefinition
     shortNames:
     - net-attach-def
  validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            config:
              type: string
```

3.2 NetworkAttachmentDefinition Object Definition

The `NetworkAttachmentDefinition` object itself contains only a "`spec`" section. Its definition (in Go form) shall be:

```
type NetworkAttachmentDefinition struct {
    metav1.TypeMeta
    // Note that ObjectMeta is mandatory, as an object
    // name is required
    metav1.ObjectMeta

    // Specification describing how to add or remove network
    // attachments for a Pod. In the absence of valid keys in
    // the Spec field, the implementation shall attach/detach an
    // implementation-known network referenced by the object's
    // name.
    // +optional
    Spec NetworkAttachmentDefinitionSpec `json:"spec"`
}


type NetworkAttachmentDefinitionSpec struct {
    // Config contains a standard JSON-encoded CNI configuration
    // or configuration list which defines the plugin chain to
    // execute. The CNI configuration may omit the 'name' field
    // which will be populated by the implementation when the
    // Config is passed to CNI delegate plugins.
    // +optional
    Config string `json:"config,omitempty"`
}
```

Plugins which are not CNI Delegating Plugins may not use the `Spec.Config` field to store arbitrary configuration. Instead they should store their non-standard configuration in annotations on the `NetworkAttachmentDefinition` or pod objects.

3.2.1 YAML Example: CNI config JSON in object

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "name": "a-bridge-network",
    "type": "bridge",
    "bridge": "br0",
    "isGateway": true,
    "ipam": {
      "type": "host-local",
      "subnet": "192.168.5.0/24",
      "dataDir": "/mnt/cluster-ipam"
    }
}'
```

3.2.2 YAML Example: CNI configlist JSON in object

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: another-bridge-network
spec:
  config: '{
  "cniVersion": "0.3.0",
  "name": "another-bridge-network",
  "plugins": [
    {
      "type": "bridge",
      "bridge": "br0",
      "ipam": {
        "type": "host-local",
        "subnet": "192.168.5.0/24"
      }
    },
    {
      "type": "port-forwarding"
    },
    {
      "type": "tuning",
      "sysctl": {
        "net.ipv4.conf.all.log_martians": "1"
      }
```

```
    }
  ]
}'
```

3.2.3 YAML Example: Limited CNI config required ("thick" plugin)

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "awesome-plugin"
}'
```

3.2.4: YAML Example: Implementation-specific Network Reference

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
```

3.3 NetworkAttachmentDefinition Object Naming Rules

Valid `NetworkAttachmentDefinition` object names must be comprised of units of the DNS-1123 label format, in accordance with how Kubernetes validates namespace names. It is recommended that each unit of DNS-1123 label be 63 characters or less.

> *"DNS-1123 label must consist of lower case alphanumeric characters or '-', and must start and end with an alphanumeric character"*
>
> - *Kubernetes Error Message*

Labels may be validated by matching this regular expression:

```
[a-z0-9]([-a-z0-9]*[a-z0-9])?
```

3.3.1 Examples

```
example-namespace/attachment-name
attachment-name
```

3.4 CNI Delegating Plugin Requirements

For implementations that are CNI Delegating Plugins, the implementation "delegates" the actual attachment/detachment to one or more additional CNI plugins. The `NetworkAttachmentDefinition` object contains the necessary information to determine which CNI plugins to execute and which options to pass to them.

3.4.1 Determining CNI Plugins for a NetworkAttachmentDefinition Object

The CNI Delegating Plugin must use the following rules, in the listed order, to determine which CNI plugin(s) to execute for a pod's attachment to a given network described by a `NetworkAttachmentDefinition` object:

1. If a "`config`" key is present in the `NetworkAttachmentDefinition.Spec` the contents of the key's value is used to execute plugins per the CNI specification
2. If a CNI .configlist file is present on-disk whose JSON "name" key matches the name of the `NetworkAttachmentDefinition` object, its contents is loaded and used to execute plugins per the CNI specification.
3. If a CNI .config file is present on-disk whose JSON "name" key matches the name of the `NetworkAttachmentDefinition` object, its contents is loaded and used to execute plugins per the CNI specification.
4. Otherwise, the network request must fail

3.4.2 Spec.Config and the CNI JSON 'name' Field

If the Spec.Config key is valid but its data omits the CNI JSON '`name`' field the CNI Delegating Plugin shall add a '`name`' field containing the `NetworkAttachmentDefinition` object's name to the CNI JSON before sending that CNI JSON configuration to a delegate plugin. This is intended to allow brevity of configuration for the "thick" plugin case where the delegate plugin requires minimal configuration.

3.4.2.1 "Name" Injection Example

Given the following `NetworkAttachmentDefinition` object:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "awesome-plugin"
```

```
  }'
```

The CNI Delegating Plugin would send the following CNI JSON Configuration to the '`awesome-plugin`' binary, generated by injecting a 'name' field with the `NetworkAttachmentDefinition` object name as its data:

```
{
    "cniVersion": "0.3.0",
    "name": "a-bridge-network",
    "type": "awesome-plugin"
}
```

3.4.3 NetworkAttachmentDefinition Object and CNI JSON Configuration Naming Considerations

Per Kubernetes requirements, each `NetworkAttachmentDefinition` object must have a name. The CNI specification strongly recommends that CNI JSON configuration include a name and future CNI specification versions will require a name.

This specification strongly recommends that the `NetworkAttachmentDefinition` object name match the `"name"` key in CNI JSON configuration referenced by the `NetworkAttachmentDefinition` object (whether stored on-disk or in the `NetworkAttachmentDefinition.Spec.Config` key). This reduces user confusion and makes the mapping between `NetworkAttachmentDefinition` object and CNI configuration clearer.

While strongly recommended, this matching is not required as it may place Kubernetes API object naming requirements on externally-defined resources. Naming is ultimately left up to the cluster administrator to decide.

4. Network Attachment Selection Annotation

To select one or more secondary ("sidecar") networks to which a pod should be attached, this specification defines a Pod object annotation. Because attachments selected by this annotation are secondary attachments, these network attachments are not known to Kubernetes itself and information about them may not be available through the standard Kubernetes API.

The Network Attachment Selection Annotation may be used to select additional attachments to the cluster-wide default network, above and beyond the required initial cluster-wide default network attachment.

4.1 Annotation Name and Format

The Pod object annotation name shall be "`k8s.v1.cni.cncf.io/networks`". The annotation value shall be specified in one of two possible formats, described below. Implementations of this specification must support both formats.

Note that even though the objects the annotation references are `NetworkAttachmentDefinition` objects, the annotation's name is "`k8s.v1.cni.cncf.io/networks`". This is intentional.

4.1.1 Comma-delimited Format

This format is intended to be a very simple, user-friendly format, composed only of `NetworkAttachmentDefinition` object references separated by commas. The format of each object reference is either (a) `<NetworkAttachmentDefinition object name>` to reference a `NetworkAttachmentDefinition` object in the pod's namespace, or (b) `<NetworkAttachmentDefinition object namespace>/<NetworkAttachmentDefinition object name>` to reference a `NetworkAttachmentDefinition` object in a different namespace.

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: net-a,net-b,other-ns/net-c
```

4.1.2 JSON List Format

This format allows users to give pod-specific requirements for the network attachment. For example, these could include a specific IP address or MAC address if those options are supported by the plugin that ultimately handles the network attachment, whether that is the implementation itself or a delegate plugin.

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {"name":"net-a"},
        {
          "name":"net-b",
          "ips": ["1.2.3.4"],
          "mac": "aa:bb:cc:dd:ee:ff"
```

```
        },
        {
          "name":"net-c",
          "namespace":"other-ns"
        }
      ]
```

4.1.2.1 JSON List Format Key Definitions

The following keys are defined for each network attachment map in this format's networks list. All key names that do not include a period character are reserved to ensure this specification may be extended in the future. Implementations that write keys other than those defined in this specification must use reverse domain name notation (eg "`org.foo.bar.key-name`") to name the non-standard keys.

4.1.2.1.1 "name"

This required key with value of type string is the name of a `NetworkAttachmentDefinition` object, either in the Pod's namespace (if the "namespace" key is missing or empty) or another namespace specified by the "namespace" key.

4.1.2.1.2 "namespace"

This optional key with value of type string is the namespace of the `NetworkAttachmentDefinition` object named by the "name" key.

4.1.2.1.3 "ips"

This optional key with value of type string-array requires the plugin handling this network attachment to assign the given IP addresses to the pod. This key's value must contain at least one array element and each element must be a valid IPv4 or IPv6 address with an optional network prefix length (eg "/24"). If the value is invalid, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation. Plugins are allowed to fail the attachment request if they require a network prefix length but the request does not contain a network prefix length. Plugins are also allowed to fail the attachment request if the specified network prefix length is unacceptable to the plugin.

4.1.2.1.3.1 "ips" Example

```
annotations:
   k8s.v1.cni.cncf.io/networks: |
    [
      {
        "name":"net-b",
        "ips": ["10.2.2.42/24", "2001:db8::5/64"]
      }
```

```
      ]
```

4.1.2.1.3.2 CNI Delegating Plugin Requirements

A "ips" key requires the CNI Delegating Plugin to add a "ips" key to the CNI "runtimeConfig" data and to set its value to a transformation of the "ips" key's value as described in CNI's CONVENTIONS.md.

The CNI Delegating Plugin must verify that at least one plugin in the attachment list supports the "ips" capability by inspecting each plugin's JSON configuration "capabilities" key. If no plugin has the "ips" capability set to true the entire attachment request must fail.

Given the immediately preceding Network Attachment Selection Annotation example, the CNI Delegating Plugin would transform the data into the following CNI JSON config snippet passed to each plugin in the CNI invocation for "net-b":

```
{
  …
  "runtimeConfig":{
    "ips": ["10.2.2.42/24", "2001:db8::5/64"]
  }
  …
}
```

4.1.2.1.4 "mac"

This optional key with value type string requires the plugin handling this network attachment to assign the given MAC address to the pod. This key's value must contain a valid 6-byte Ethernet MAC address or a valid 20-byte IP-over-InfiniBand Hardware address (as described in RFC4391 section 9.1.1). If the value is invalid, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation.

4.1.2.1.4.1 "mac" Example

Given the following Network Attachment Selection Annotation for a pod:

```
  annotations:
     k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name":"net-b",
          "mac": "02:23:45:67:89:01"
        }
      ]
```

4.1.2.1.4.2 CNI Delegating Plugin Requirements

A "mac" key requires the CNI Delegating Plugin to add a "mac" key to the CNI "runtimeConfig" data and to set its value to a transformation of the "mac" key's value as described in CNI's CONVENTIONS.md.

The CNI Delegating Plugin must verify that at least one plugin in the attachment list supports the "mac" capability by inspecting each plugin's JSON configuration "capability" key. If no plugin has the "mac" capability set to true the entire attachment request must fail.

Given the immediately preceding Network Attachment Selection Annotation example, the CNI Delegating Plugin would transform the data into the following CNI JSON config snippet passed to each plugin in the CNI invocation for "net-b":

```
{
  …
  "runtimeConfig":{
    "mac": "02:23:45:67:89:01"
  }
  …
}
```

4.1.2.1.5 "interface"

This optional key with value of type string requires the implementation to use the given name for the pod interface resulting from this network attachment. This key's value must be a valid Linux kernel interface name. If the value is invalid, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation.

If the requested interface name is already used by a previous network attachment, the implementation must fail the current network attachment.

4.1.2.1.5.1 CNI Delegating Plugin Requirements

An "interface" key requires the CNI Delegating Plugin to set the CNI_IFNAME environment variable to the given value when invoking CNI plugins for this network attachment.

4.1.2.1.6 "cni-args"

This optional key with value of type dict requires CNI delegating plugins to pass the value through to the child plugin. CNI Delegating Plugins must implement the "cni-args" key as described below, however, other implementations may ignore this key.

4.1.2.1.6.1 "cni-args" Example

```
  annotations:
    k8s.v1.cni.cncf.io/networks: |
```

```
[
  {
    "name":"net-b",
    "cni-args": {
       "spoofchk": "on"
    }
  }
]
```

4.1.2.1.6.2 CNI Delegating Plugin Requirements

A “cni-args” key requires the CNI Delegating Plugin to add the "args" key to the CNI JSON configuration passed to the child plugin as defined in CONVENTION.md. To synthesize the final “args” value passed to the child plugin, the implementation must combine the values from the following sources in the following priority order:

- "cni-args" key’s in Network Attachment Selection Annotation of the Pod
- The "args" key’s value in Network Attachment Definition Spec.Config CNI configuration JSON

4.1.2.1.7 "portMappings"

This optional key with value of type dict-array requires the implementation to configure traffic redirection from a IP port bound to the host’s IP address on this network attachment’s network to a container IP port bound to the pod interface resulting from this network attachment. If the value is invalid, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation.

This key’s value must contain at least one array element and each element must be a valid dict containing two or more of the following keys:

- “hostPort”: this required key of type integer must contain a value between 1 and 65535 inclusive. It specifies the port number on the host on which to accept traffic and redirect that traffic to the container port.
- “containerPort”: this required key of type integer must contain a value between 1 and 65535 inclusive. It specifies the port on which the container expects to accept traffic that is redirected from the host port.
- “protocol”: this optional, case-insensitive key of type string must contain one of “UDP”, “TCP”, or “SCTP”. It specifies the protocol of traffic to redirect. If not specified, this value defaults to "TCP".

If the underlying network does not support port mappings (for example, because the underlying network is not available from the host) the attachment request must fail.

4.1.2.1.7.1 "portMappings" Example

```
annotations:
   k8s.v1.cni.cncf.io/networks: |
    [
      {
        "name":"net-b",
        "portMappings": [
          { "hostPort": 8080, "containerPort": 80, "protocol": "tcp" }
        ]
      }
    ]
```

4.1.2.1.7.2 CNI Delegating Plugin Requirements

A "portMappings" key requires the CNI Delegating Plugin to add a "portMappings" key to the CNI "runtimeConfig" data and to set its value to a transformation of the "portMappings" key's value as described in CNI's CONVENTIONS.md.

The CNI Delegating Plugin must verify that at least one plugin in the attachment list supports port mapping by inspecting each plugin's JSON configuration "capabilities" key. If no plugin has the "portMappings" capability set to true the entire attachment request must fail.

4.1.2.1.8 "bandwidth"

This optional key with value of type dict requires the implementation to configure bandwidth management on this network attachment. If the value is invalid, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation. Implementations may choose any bandwidth management mechanism that allows ingress and egress bandwidth limiting, and are free to use a different mechanism for ingress and egress.

This key's value must contain at least one dict element and each element must be one of the following key/value pairs:

- "ingressRate": this optional key of type integer must contain a non-zero value that specifies the maximum ingress bandwidth rate in bits-per-second.
- "ingressBurst": this optional key of type integer must contain a non-zero-value that specifies the maximum allowed burst in bits. If "ingressBurst" is present, "ingressRate" must also be present for ingress bandwidth management to be valid.
- "egressRate": this optional key of type integer must contain a non-zero value that specifies the maximum egress bandwidth rate in bits-per-second.
- "egressBurst": this optional key of type integer must contain a non-zero value that specifies the maximum allowed burst in bits. If "egressBurst" is present, then "egressRate" must also be present for egress bandwidth management to be valid.

If the underlying network does not support bandwidth management the attachment request must fail.

4.1.2.1.8.1 "bandwidth" Example

```
annotations:
   k8s.v1.cni.cncf.io/networks: |
    [
      {
        "name":"net-b",
        "bandwidth: {
          "ingressRate": 2048, // Limit ingress to 2Kbps
          "ingressBurst": 300, // Allow small egress bursts of 300b
          "egressRate": 8000,  // Limit egress to 8000bps
          "egressBurst": 200   // Allow small egress bursts of 200b
        }
      }
    ]
```

4.1.2.1.8.2 CNI Delegating Plugin Requirements

A "bandwidth" key requires the CNI Delegating Plugin to add a "bandwidth" key to the CNI "runtimeConfig" data and to set its value to a transformation of the "bandwidth" key's value as described in CNI's CONVENTIONS.md. Implementations may wish to choose reasonable default values for "ingressBurst" and "egressBurst" as some bandwidth management plugins (like the CNI reference "bandwidth" plugin) require these values.

The CNI Delegating Plugin must verify that at least one plugin in the attachment list supports traffic shaping by inspecting each plugin's JSON configuration "capabilities" key. If no plugin has the "bandwidth" capability set to true the entire attachment request must fail.

4.1.2.1.9 "default-route" Default route selection for a particular attachment

This optional key with value of type string-array is used to explicitly select which attachment will receive the default route. The value of items in the "default-route" array are intended to be gateways, e.g. an IP address to which packets that do not match any other routes are sent. This key must only be set on one item in the Network Attachment Selection Annotation. This list may be empty.

Typically, it's assumed that the attachment for the default network will have the default route, however, in some cases one may desire to specify which attachment will have the default route. When "default-route" is set for an attachment other than the cluster-wide default network attachment, it should be noted that the default route and gateway will be cleared from the cluster-wide default network attachment.

Only one network attachment may specify the "default-route" key. If more than one attachment specifies the key, the Network Attachment Selection Annotation shall be invalid and ignored by the implementation.

It is the responsibility of the implementation to determine how routing will be manipulated. For example, a CNI Delegating Plugin might inject a CNI IPAM plugin which can manipulate routes, and will flush the default route from the default network attachment and any attachments that do not have the “default-route” key set, and then it will set the default route.

When multiple items appear in the gateway list, order of those items may matter. An implementation may choose to assign a metric to each gateway in the list using that ordering.

4.1.2.1.9.1 “default-route” Example

In this example, the default route will be set on “net-b” and traffic routed to the gateway as specified in the “default-route” key.

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        { "name":"net-a" },
        {
          "name":"net-b",
          "default-route": ["10.0.0.1"]
        }
      ]
```

4.2 Multiple Attachments to the Same Network

Pods may request attachment to the same network multiple times via the Network Attachment Selection Annotation. Each of these requests is considered a separate attachment, must be processed by the implementation as a separate operation, and must result in a separate Network Attachment Status Annotation entry.

4.2.1 CNI Delegating Plugin Requirements

Each network reference in the Network Attachment Selection Annotation corresponds to a CNI plugin/configlist invocation described by a `NetworkAttachmentDefinition` object. As the CNI specification defines a network attachment as a unique tuple of `[container ID, network name,CNI_IFNAME]`, the CNI Delegating Plugin must ensure that all CNI operations (eg ADD or DEL) for a given network attachment use the same unique tuple, which should be created as follows:

1. Container ID: given by the runtime
2. Network Name: present in (or found via) the NetworkAttachmentDefinition object

3. CNI_IFNAME: if not otherwise specified by the Network Attachment Selection Annotation, generated by the CNI Delegating Plugin for a given attachment, and must be unique across all attachments for the given [Container ID, network name] tuple.

4.2.2 Example

```
annotations:
  k8s.v1.cni.cncf.io/networks: net-a,net-a
```

In this example the implementation must attach "net-a" twice and each attachment would result in a separate entry in the Network Attachment Status Annotation list.

5. Network Attachment Status Annotation

To ensure that the result of a network attachment is available via the Kubernetes API, the implementation may publish the result of the attachment operation to an annotation on the pod object that requested the attachment.

The annotation's name shall be "`k8s.v1.cni.cncf.io/network-status`" and its value shall be a JSON-encoded list of maps. Each element in the list shall be a map composed from the result of a network attachment operation as described below.

5.1 Source of Status Information

A network attachment operation's status map shall contain information taken from the result of the attachment operation for the given network. Status information that is only useful inside the pod itself (like IP routes) does not need to be part of the status map as this information is not generally relevant to Kubernetes API clients.

5.1.1 CNI Delegating Plugin Requirements

Status for the attachment shall be taken from the first sandbox interface of the CNI `Result` object for that attachment's CNI ADD or GET invocation. If the ADD or GET invocation does not return a result (which is currently allowed by the CNI specification) the implementation should add a minimal status map as described below.

The implementation should use as much information provided in the CNI Result object as possible to construct the Network Attachment Status Annotation. If the operation returns a CNI Result object of version 0.3.0 or later, the "interfaces", "ips", and "dns" fields can be used to easily construct the Network Attachment Status Annotation. If the operation returns a CNI Result object less than version 0.3.0, the implementation should construct as much of the Network Attachment Status Annotation as it can from the CNI Result, and may populate the remaining Annotation fields (eg 'interface' and 'mac') by whatever means it wishes to.

5.2 Cluster-Wide Default Network Entry

Since the implementation is required to attach the pod to the cluster-wide default network, this network must have an entry in the status map even though it is not specified in the Network Attachment Selection Annotation and may not have a corresponding `NetworkAttachmentDefinition` object. The entry shall set its '`default`' key to '`true`'. The default entry may be at any location in the status list.

5.3 Status Map Key Definitions

The following keys are defined for each network attachment's status map. All key names that do not include a period character are reserved to ensure this specification may be extended in the future. Implementations that write keys other than those defined in this specification must use reverse domain name notation (eg "`org.foo.bar.key-name`") to name the non-standard keys.

5.3.1 "name"

This required key's value (type string) shall contain either a `NetworkAttachmentDefinition` object name from the pod's Network Attachment Selection Annotation, or the name of the cluster-wide default network. The "name" may contain a namespace reference as defined in Section 4.1.1. [ref 2018-02-01 meeting @22:30]

5.3.2 "interface"

This optional key's value (type string) shall contain the network interface name in the pod's network namespace corresponding to the network attachment.

5.3.2.1 CNI Delegating Plugin Requirements

For a CNI Result of version 0.3.0 or later, the "interface" key should be sourced from the first element of the CNI Result's "interfaces" property which has a valid "sandbox" property.

For a CNI Result version earlier than 0.3.0, the implementation may populate the field from the CNI_IFNAME environment variable it set for the network attachment operation, or leave the field blank.

5.3.3 "ips"

This optional key's value (type string array) shall contain an array of IPv4 and/or IPv6 addresses assigned to the pod as a result of the attachment operation.

5.3.3.1 CNI Delegating Plugin Requirements

For a CNI Result of version 0.3.0 or later, the "ips" key should be sourced from the CNI Result's "ips" property if either of the following are true; otherwise it should not be included in the status map.

1. The "ips" key should be taken from the element of the CNI Result object's "ips" list where the "interface" index refers to the first element in the CNI Result object's "interfaces" list with a valid "sandbox" key.
2. If there are no elements in the CNI Result object's "interfaces" list, or none of the interfaces in the CNI Result object's "interfaces" list have valid "sandbox" properties, the "ips" key should be taken from the first element of the CNI Result object's "ips" list which does not have an "interface" property or where the "interface" property is less than zero.

The object of these requirements is to ensure that the IP address included in the status map is the same IP address assigned to the attachment's interface inside the pod, not the address of non-sandbox interfaces which are sometimes reported by CNI plugins.

For a CNI Result of a version earlier than 0.3.0, the "ips" key should be sourced from a combination of the CNI Result's "ip4" and "ip6" properties.

5.3.4 "mac"

This optional key's value (type string) shall contain the hardware address of the network interface named by the "interface" key. If the "mac" key is present, "interface" must also be present.

5.3.4.1 CNI Delegating Plugin Requirements

For a CNI Result of version 0.3.0 or later, the "mac" key should be sourced from the first element of the CNI Result's "interfaces" property which has a valid "sandbox" property.

For a CNI Result version earlier than 0.3.0, the implementation may populate the field by an implementation-specific mechanism, or leave the field blank.

5.3.5 "default"

This required key's value (type boolean) shall indicate that this attachment is the result of the cluster-wide default network. Only one element in the Network Attachment Status Annotation list may have the "default" key set to true.

5.3.6 "dns"

This optional key's value (type map) shall contain DNS information that is gathered as a result of the network attachment. The map may contain the following keys.

5.3.6.1 "nameservers"

This optional key's value (type string array) shall contain an array of IPv4 and/or IPv6 addresses of DNS servers.

5.3.6.2 "domain"

This optional key's value (type string) shall contain the local domain name for the network attachment.

5.3.6.3 "search"
This optional key's value (type string array) shall contain the array of DNS search names for the network attachment.

5.3.6.4 "default-route"
This optional key's value (type string array) shall be set to the default gateway (or default gateways in the case of IPv6 with multiple default routes) for the network attachment, if the attachment had a "default-route" key in the Network Attachment Selection Annotation.

5.4 Example

With a single gateway:

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/network-status: |
      [
        {
          "name": "cluster-wide-default",
          "interface": "eth5",
          "ips": [ "192.0.2.2/24", "2001:db8::2230/32" ],
          "mac": "02:11:22:33:44:54",
          "default-route": ["192.0.2.1"],
          "default": true
        },
        {
          "name": "some-network",
          "interface": "eth1",
          "ips": [ "192.0.2.2/24", "2001:db8::2234/32" ],
          "mac": "02:11:22:33:44:55",
          "dns": {
            "nameservers": [ "192.0.2.1", "2001:db8:4860::8888" ],
            "search": [ "eng.example.com", "example.com" ]
          }
        },
        {
          "name": "other-ns/an-ip-over-infiniband-network",
          "interface": "ib0",
```

```
          "ips": [ "198.51.100.1/24" ],
          "mac":
"80:00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:00:11:22"
        }
      ]
```

With multiple gateways:

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.io/network-status: |
      [
        {
          "name": "cluster-wide-default",
          "interface": "eth5",
          "ips": [ "192.0.2.2/24", "2001:db8::2230/32" ],
          "mac": "02:11:22:33:44:54",
          "default-route":["192.0.2.1","2001:db8::1","2001:db8::2"],
          "default": true
        },
        {
          "name": "some-network",
          "interface": "eth1",
          "ips": [ "192.0.2.2/24", "2001:db8::2234/32" ],
          "mac": "02:11:22:33:44:55",
          "dns": {
            "nameservers": [ "192.0.2.1", "2001:db8:4860::8888" ],
            "search": [ "eng.example.com", "example.com" ]
          }
        },
        {
          "name": "other-ns/an-ip-over-infiniband-network",
          "interface": "ib0",
          "ips": [ "198.51.100.1/24" ],
          "mac":
"80:00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:00:11:22"
        }
      ]
```

6. Cluster-Wide Default Network

The implementation must attach every pod to the cluster-wide default network, keeping the existing Kubernetes pod networking behavior. [ref 2018-01-18@31:30] The implementation is free to define how the cluster-wide default network is configured and referenced (for example, as a `NetworkAttachmentDefinition` object, an on-disk CNI JSON configuration file, or some other means), provided that all pods are first attached to this network.

It's generally expected that the the cluster-wide default network attachment sets a default route. This default route may be overridden using the "default-route" key in the Network Attachment Selection Annotation (see 4.1.2.1.9). Notably, if the cluster-wide default network attachment does not set a default route and the "default-route" key has not been set, this may cause pods which do not route traffic in a manner which is typically expected (that is, to other pods connected to the default network).

6.1 Network Plugin Readiness

The implementation should not write its CNI JSON configuration file to the Kubernetes CNI configuration directory until the cluster-wide default network is ready. This prevents Kubernetes from scheduling pods on the node that will immediately fail because the implementation has signalled readiness but the cluster-wide default network has not.

The implementation is free to define how cluster-wide default network readiness is determined, provided that when the cluster-wide default network is determined to be ready, a pod may be immediately attached to that network and have a reasonable expectation of network connectivity.

6.1.1 CNI Delegating Plugin Recommendations

To prevent race conditions between the cluster-wide default network, the implementation, and kubelet's CNI plugin loader, it is recommended that kubelet be configured with an implementation-specific CNI configuration directory through the `--cni-conf-dir` command-line option. The implementation should wait until the cluster-wide default network plugin's CNI JSON configuration is written to `/etc/cni/net.d` and then write its own CNI JSON configuration file to the implementation-specific CNI configuration directory given to kubelet.

6.1.2 Alternate Readiness Method

If it is not possible to wait for the cluster-wide default network to be ready before indicating the implementation's readiness to kubelet, the implementation may immediately install its own CNI configuration file to the kubelet CNI configuration directory, ensuring its configuration takes precedence over the cluster-wide default network's (if any). The implementation must then block any pod network attachment/detachment operations until the cluster-wide default network is ready.

6.2 Cluster-wide Default Network Attachment Ordering

The implementation must attach the cluster-wide default network before attaching any networks specified by the Network Attachment Selection Annotation.

7. Runtime and Implementation Considerations

7.1 CNI Delegating Plugin Requirements for CNI Configuration and Result Versioning

The CNI Delegating Plugin must conform to the CNI specification's requirements for invoking plugins in a CNI configlist. If the CNI Delegating Plugin uses the CNI project's reference "libcni" library these issues will be handled automatically. If not, the CNI specification requires that the CNI Delegating Plugin inject the `cniVersion` and `name` fields from the configlist into each plugin invocation's configuration JSON. This ensures that each plugin in the configlist is able to understand the result of the previous plugin, and that the runtime receives the correctly-versioned final result.

7.2 Attachment/Detachment Failure Handling

On pod network setup, the failure to attach any network referenced by a pod's Network Attachment Selection Annotation or the failure to attach the cluster-wide default network shall immediately fail the pod network setup operation. Attachments that have not yet been performed shall not be attempted.

On pod network teardown, all network attachments that were attempted during pod network setup must be torn down, and the failure of one shall not prevent teardown of subsequent attachments. However, if any detachment failed, a final error shall be delivered to the runtime to indicate the overall teardown operation failed.

7.3 Serialization of Network Attachment Operations

The CNI Specification 0.4.0 states "The container runtime must not invoke parallel operations for the same container, but is allowed to invoke parallel operations for different containers." Implementations must follow this requirement and must not parallelize pod network attachment operations. This requirement may be lifted in future versions of the specification.

7.4 Restrictions on Selection of Network Attachment Definitions by Pods

Implementations are free to impose restrictions on which Network Attachment Definitions can be selected by a given Pod. This could be done by RBAC, admission control, or any other implementation-specific method. If the implementation determines that a given Network Attachment Definition selected by the Pod is not allowed for that Pod, it must fail the pod network operation.

7.5 "runtimeConfig" Handling

Any "runtimeConfig" options passed from Kubernetes to the implementation via CNI JSON configuration must not be passed to or applied by network attachments other than the

cluster-wide default network. Kubernetes expects these options to apply to only the cluster-wide default network and they may not be generally applicable.

Open Questions

1. Security
    - Should a cluster or namespace administrator (or somebody else) be able to prevent certain pods from attaching to certain networks?
        - PodSecurityPolicy: https://kubernetes.io/docs/concepts/policy/pod-security-policy/
        - RBAC
2. Dynamic attachment updates
    - Add/remove sidecar networks while the pod is running (since the sidecars are not reported to Kubernetes, there is issue with changing the Pod's Kube API IP address at runtime)
    - Requires a long-running meta-plugin to listen for changes on Pod objects and reconcile the pod's attachments
    - What happens if a dynamic attachment fails? Does that kill the pod entirely, or is that error information propagated to the status map or a Pod Event?

Previously Discussed Topics

1. Default-ness (which network gets the default route, which network is reported to Kube, etc)
    a. Decision: always attach the default Kube network, and all networks specified by this CRD are sidecars which are not reported to Kube
        i. https://youtu.be/_iK-DNJmzY0?list=PL69nYSiGNLP2E8vmnqo5MwPOY25sDWIxb&t=1885 (31:30)
    b. Previously considered, punted for future:
        i. allow not attaching default Kube network
        ii. which network gets passed back to Kube
        iii. which network gets the pod default route
        iv. which network is used for health checking
2. Attachment "pairing" objects instead of Pod annotation of Network names (Mike)
    a. Decision: stick with annotation on pod for now
    b. Would support selectors to determine which pod and attachment factory to use
    c. Allows for more easily finding all attachments that apply to a given factory (rather than scanning all pods to find ones which have the right annotation)
    d. Would allow for pod-specific details (like static IP address or other properties) without adding more annotations to the Pod object
    e. Downside: more user/admin work to connect pods to networks, user would have to define the attachment factory, and then for each pod also define an attachment object. Could be automated of course…
3. Sidecar Network Interaction with Services

    a. Decision: networks explicitly selected by this mechanism have no defined interaction with Kubernetes services or any other part of the Kube API, with the exception of the network status annotation added to pods. It is up to the plugin whether or not it wants to provide additional interaction with the Kube API (eg, allows Services to be used on the sidecar networks or whatever)
4. Overlapping IP networks
    a. Decision: punted to network plugins to manage; either the plugin does something specific and well defined, or just Don't Do This.
    b. Per advocates for a more well-defined way to handle overlapping network attachments
5. Networks as Resources
    a. Decision: v1 of this specification will not address this question. It was discussed heavily in the 2018-03-01 meeting [video] [minutes] and we agreed there were definitely interesting ideas that should be explored but that the current proposals were not adequate.
6. "Overrides" as part of the JSON-style annotation
    a. Decision: Likely target for v2, not a target v1
    b. General idea:
        i. The possibility of an "overrides" variable. So, let's say it's bridge type plugin. One sets:
        ii. `"overrides": { "hairpin": true }
        iii. The metaplugin checks the overrides against the configuration it loads from CRD objects against what's provided in overrides, it sees that a key is set in "overrides" and it overwrites the key (or sets it if unset) in the configuration. I'd say for simplicity it would be limited to top-level key.
    c. One commenter replied given an NFV use case, at a high level: *"actually this is a strong NFVI/TelCo request to be able to drive IP management from the Pod level"*